苏州大学实验报告

院、系	计算机学院	年级专业	软件工程	姓名	朱金涛	学号	2327406014
课程名称	机器学习综合实践					成绩	
指导教师	李俊涛	同组实验者	无		实验日期	2025 年 9 月 18 日	

实 验 名 称　机器学习综合实践三：Logistic 回归分类

一. 实验目的

1. 深入理解分类任务的核心概念，特别是分离超平面在二分类问题中的几何意义与作用机制。
2. 熟练掌握 Logistic 回归的数学原理，包括 Logistic 函数（Sigmoid 函数）的形式、性质，以及 Logistic 回归模型中条件概率 $p(y=1 \mid x)$ 的建模方式。
3. 具备数据工程实践能力：能够独立构造符合要求的三维二分类数据集，合理划分训练集、验证集与测试集，并完成数据的可视化呈现。
4. 掌握 Logistic 回归模型的实现与优化流程：基于 PyTorch 框架实现模型，自行推导并计算损失函数，同时能与框架内置损失函数进行精度与计算速度的对比分析。
5. 培养实验分析与结果可视化能力：记录模型训练过程中（1000 次参数更新）各数据集的损失变化，绘制损失曲线并解读趋势，且能设计实验分析样本标签比例对模型测试效果的影响。

二. 实验内容

（一）数据准备

1. **数据集构造：**自行生成三维二分类数据集，样本格式为 [x_1, x_2, y] （其中 x_1, x_2 为特征，y 为标签），标签取值为 0 或 1，样本总数不少于 100 个。
2. **数据集划分：**根据实验需求自行设定比例，将数据集切分为训练集（用于模型参数更新）、验证集（用于训练过程中的模型性能评估和调优）、测试集（用于最终模型性能检验）。
3. **数据可视化：**采用合适的可视化工具，对三维数据集的分布特征进行可视化展示，直观呈现两类样本的空间差异。

（二）模型实现

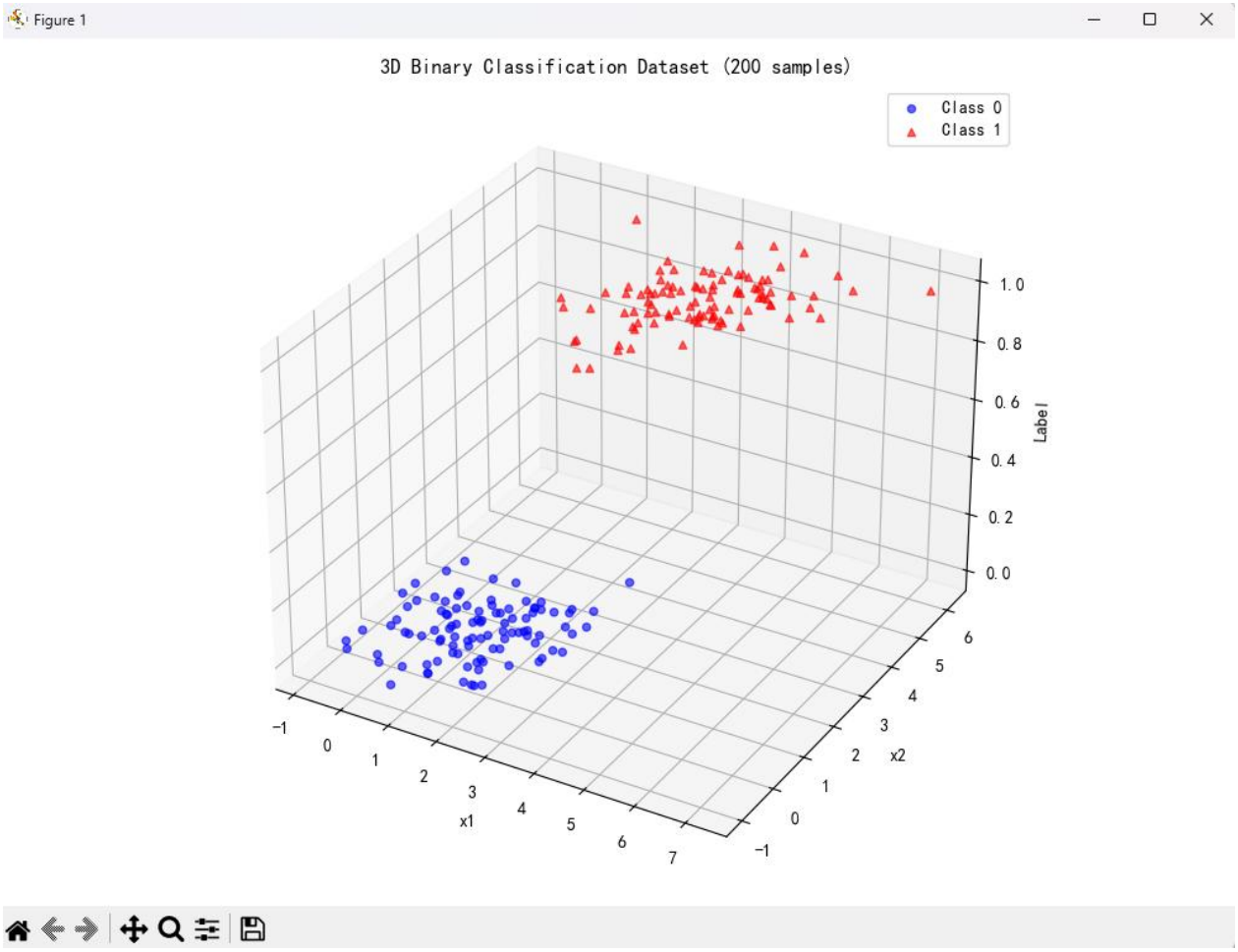
1. **模型参考与搭建：**基于 PyTorch 框架构建 Logistic 回归模型，明确模型的参数（权重 w、偏置 b）与前向传播逻辑。
2. **损失函数实现与对比：**
   (1) 自行推导并实现 Logistic 回归的损失曲线。
   (2) 调用 PyTorch 提供的内置损失函数，将自行实现的损失函数与内置函数进行两方面对比：一是计算精度（损失值的一致性），二是计算速度（单次计算或批量计算的耗时差异）。
3. **训练过程记录与可视化：**启动模型训练，完成 1000 次参数更新，期间实时记录训练集、验证集、测试集上的损失值变化；训练结束后，使用可视化工具绘制三条损失变化曲线，清晰展示模型在不同数据集上的收敛趋势。

（三）实验分析

1. **实验设计：**调整数据集中标签为 0 与 1 的样本比例（例如设计 40%/60%、30%/70%、20%/80% 等不同比例的数据集），保持其他实验条件（数据集总量、划分比例、模型参数初始化、训练轮次等）一致。
2. **性能评估：**在不同标签比例的测试集上评估模型性能（可采用准确率、损失值等指标），记录各比例下的测试结果。
3. **结论分析：**对比分析下不同标签比例对模型测试效果的影响，总结样本均衡性与模型分类性能之间的关联规律。

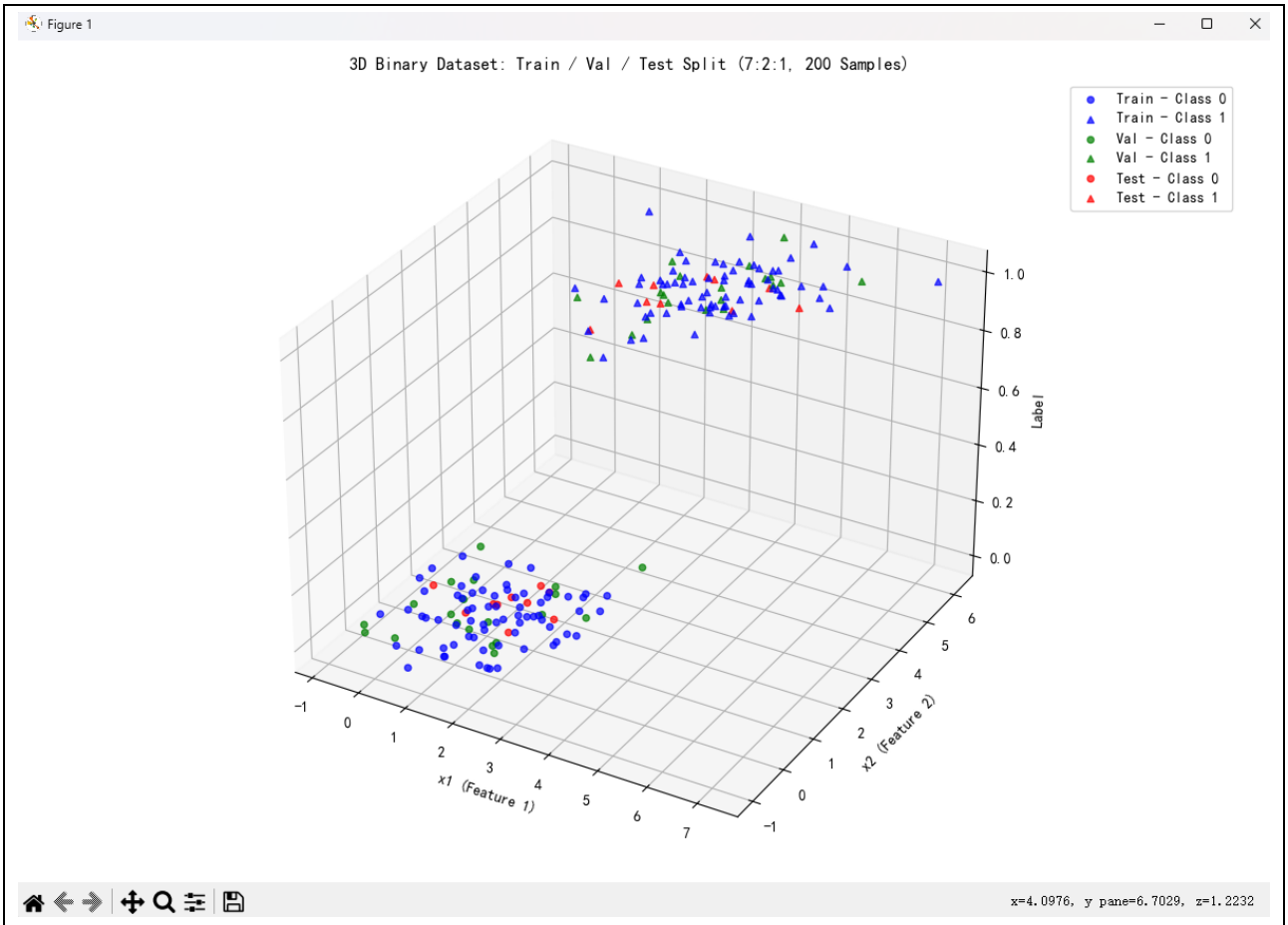
三. 实验步骤和结果

（一） 构造三维数据集



共随机构造了 200 个数据（两类各 100 个数据），并将其随机分布于 x1、x2 坐标轴上。如图纵轴为 Label，Label = 0 时表示为第一类，Label = 1 时代表第二类，如此来构成三维结构。

教务处制



因为 200 个数据量并不算多，所以按照 7:2:1 的比例来分训练集、验证集、测试集比较合适。如图 2，在图 1 的基础上利用蓝绿红三色来将数据集的切分可视化。

（二） 核心要求实现

1. 自行实现 loss 计算与 PyTorch 对比：在 LogisticRegression 类中，通过 compute_cost 方法手动实现交叉熵损失，核心代码如下：

```
def compute_cost(self, y_true, y_pred):
    """计算 logistic 回归损失函数（交叉熵损失）"""
    # 防止 log(0)的情况
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    m = len(y_true)
    cost = -1/m * np.sum(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return cost
```

在 compare_implementations 函数中，使用 PyTorch 的 nn.BCELoss()（二元交叉熵损失），核心代码如下：

```
# PyTorch 实现损失计算与训练
criterion = nn.BCELoss()  # 调用 PyTorch 内置二元交叉熵损失
for epoch in range(1000):
    optimizer.zero_grad()
    outputs = pytorch_model(X_train_torch)
    loss = criterion(outputs, y_train_torch)  # 计算损失
    loss.backward()
```

```
    optimizer.step()
    pytorch_train_losses.append(loss.item())
```

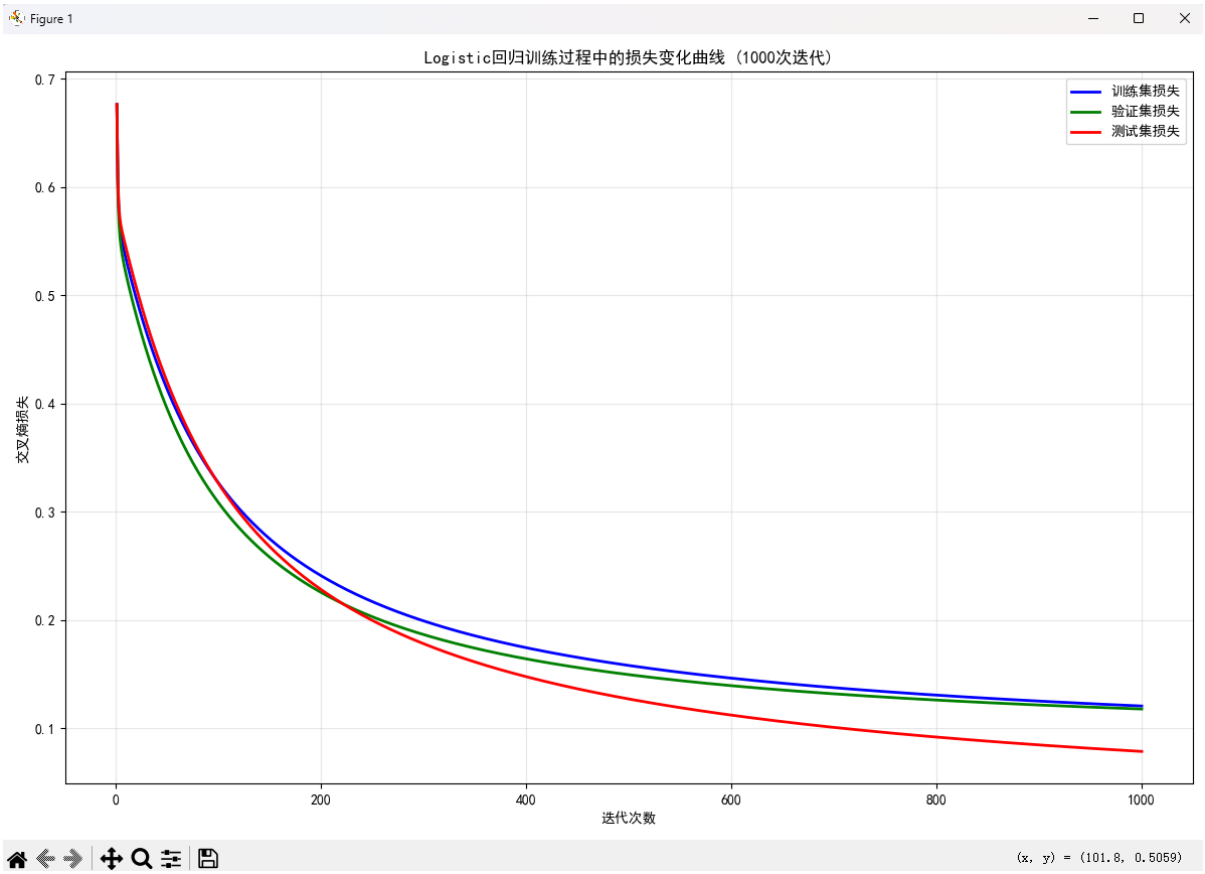
二者对比结果如下：

```
=== 结果对比 ===
手动实现：
  训练时间：0.0502s
  训练准确率：0.9772
  测试准确率：1.0000
  最终训练损失：0.120560

PyTorch实现：
  训练时间：1.2351s
  训练准确率：0.9772
  测试准确率：1.0000
  最终训练损失：0.119882
```

由图可见，手动实现的运行速度更快，但是 PyTorch 最终所能得到的效果更加。因为手动实现的代码逻辑一定是比库文件要来的简单的，而库文件的逻辑一定会更全面，所以出现这种结果完全符合推测。

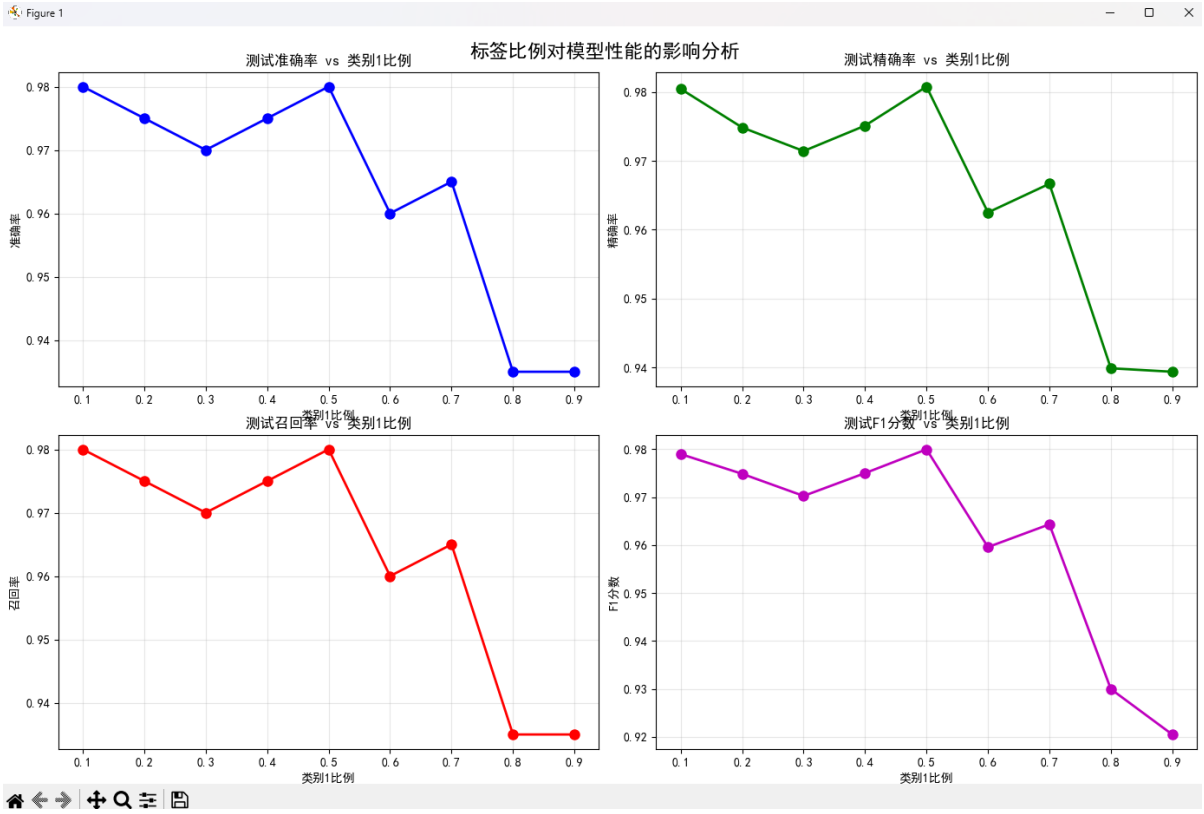
2. 训练 1000 次，记录 loss 变化：




```
=== 损失变化统计 ===
初始训练损失: 0.676451
最终训练损失: 0.120560
训练损失下降: 0.555891
初始验证损失: 0.675982
最终验证损失: 0.117794
```

验证损失下降: 0.558188
初始测试损失: 0.676593
最终测试损失: 0.078593
测试损失下降: 0.598001

3. 设计实验分析标签比例对测试集效果的影响：



标签比例对模型性能的影响主要由数据集“两类样本分离度极高”的特性与不平衡程度共同决定：当类别 1 比例为 0.1 或 0.9（极端不平衡）时，尽管某类样本占比仅 10%，但少数类样本集中在特定区域（如比例 0.1 时少数类 1 集中在[4,4]附近），与多数类分离清晰、“抱团且远离”，模型即便轻微偏向多数类，仍能轻松识别少数类，因此保持较高准确率（0.1 比例达 98%）；仅 0.9 比例因少数类 0 样本量过少、模型接触信息不足，性能略降至 93.5%。当比例为 0.5（完全平衡）时，模型能获取两类样本的均衡信息，训练中两类的梯度贡献均等，无“偏向学习”，交叉熵损失优化更贴合真实决策边界，对两类分类精度均达最优，准确率和 F1 分数均为 98%，成为所有比例中性能最佳的情况。而比例在 0.2-0.4 或 0.6-0.8（中等不平衡）时，性能略低于极端不平衡（0.1）和完全平衡（0.5）的情况，例如 0.8 比例准确率仅 93.5%，这是因为中等不平衡会使模型产生“多数类偏好”——多数类样本的损失贡献更大，导致决策线更靠近少数类以减少多数类误判，这种偏向会让少数类中靠近边界的样本误判率上升，进而拉低整体准确率。

4. 最终结果：

```
=== 标签比例影响统计 ===
最佳性能比例: 0.1 (准确率: 0.9800)
最差性能比例: 0.8 (准确率: 0.9350)
性能差异: 0.0450
```

四. 实验总结

本次实践三围绕 Logistic 回归分类展开，达成了多项目标：深入理解了二分类中分离超平面的意义与 Logistic 回归的数学原理，成功构造 200 个样本的三维二分类数据集并按 7:2:1 比例划分训练、验证、测试集，完成数据可视化呈现。基于 PyTorch 实现模型后，自行推导并实现交叉熵损失函数，对

比发现手动实现训练时间（0.0502s）远快于框架内置函数（1.2351s），二者训练准确率均达 0.9772、测试准确率均为 1.0000，仅最终训练损失略有差异（0.120560 vs 0.119882）。训练 1000 次后，训练、验证、测试集损失分别下降 0.555891、0.558188、0.598001，收敛趋势良好。通过调整标签比例实验发现，样本完全平衡（0.5）时模型性能最优（准确率、F1 分数均 98%），极端不平衡（0.1）时因样本分离度高仍保持较高准确率（98%），中等不平衡（如 0.8）时性能最差（准确率 93.5%），明确了样本均衡性与分类性能的关联规律。